PCT/EP2004/050564

19. 04. 2004

Europäisches Patentamt
European Patent Office
Office européen des brevets

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

03290954.1

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Anmeldung Nr:
Application no.: 03290954.1
Demande no:

Anmeldetag:
Date of filing: 17.04.03
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

Thomson Licensing, Inc.
46, Quai Alphonse Le Gallo
92100 Boulogne-Billancourt
FRANCE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Data requesting and transmitting devices and processes and corresponding products

In Anspruch genommene Priorität(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

H04N7/173

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL PT RO SE SI SK TR LI

Data requesting and transmitting devices and processes and corresponding products

The present invention relates, among other things and without restriction, to data requesting and transmitting devices and processes, notably for VOD (for "Video On Demand"), as well as to corresponding products.

According to existing techniques available for PCs (for "Personal Computers"), it is possible to order video and/or audio data (hereinafter noted "AV data") from a server through the Internet, by connecting a requesting PC to the server and giving thereto the IP (for "Internet Protocol") address of the PC. This is applied for example to trailers or advertisements. Those techniques, which rely on streaming, enable not only to download AV data in storing spaces for later displaying, but also to get the AV data at the same time as they are displayed.

However, such achievements require correspondingly high local resources at the PCs, in terms of CPU (for "Central Processing Unit") processing and of memory, to store and exploit suitably the received data. Moreover, control operations on the streamed AV data are very limited, and do not comprise for example pausing with later resuming, or slow or accelerated motion freely triggered or adjusted by a user during playing.

It is also known to obtain AV data from a server with a PC through a local network, by reading selected information stored in the server and transferring it in a buffer of the PC, and by playing the AV data accumulated in the buffer at the same time the data continue to be read, as soon as the buffer contains enough information (typically a few seconds).

Such a technique is only available for networks of limited sizes, and is notably not appropriate for WANs (for "Wide Area Networks"), since the

correct playing of AV data by a PC requires regular and fast access to the server in which they are stored.

The present invention concerns a data requesting device enabling flexible obtaining of data, notably AV (Audio/Video) data, with possibly limited required resources in terms of processing, memory and storing, included from a remote server through a WAN. The requesting device of the invention may notably allow to pause and retrieve a movie being played from such a server.

The invention also concerns a data requesting process, data transmitting device and process, and products, corresponding to that requesting device and able to offer similar advantages.

To that effect, the invention is related to devices, processes and products as defined in the claims.

Surprisingly, the invention combines pull and push steps, alternating them to obtain the wished information in the form of successively required portions. This contrasts with the known solution in which either a purely push mode is used (streaming from a server through Internet) or a purely pull mode is operated (communication with a server through a local network).

The invention will be better understood and illustrated by means of the no limitative following examples of embodiments and of putting in practice, with reference to the appended figures on which:

- Figure 1 shows an communication environment involving notably requesting devices incorporated in IRDs (for "Integrated Receivers Decoders") and a transmitting device incorporated in a server, according to the invention;
- Figure 2 represents pipes and buffers involved in streaming in the communication environment of Figure 1;

- Figure 3 is a protocol state transition diagram in the communication environment of Figure 1, from server side;

- Figure 4 is a streaming state transition diagram in the communication environment of Figure 1, from server side;

- Figure 5 represents a processing overview in the IRDs of Figure 1;

- and Figure 6 illustrates functionalities of an injector in one of the IRDs of Figure 1.

A communication environment (Figure 1) comprising a server and IRDs (IRD1... IRDn) communicating through PSTN wires provides a real video streaming system fitting on the Internet world. The server is connected to a first VDSL modem (for "Very high speed Digital Subscriber Loop") by means of an Ethernet link, while the IRDs are for instance respectively associated with TV sets and are connected to a second VDSL modem via a switch, through an Ethernet link.

The VDSL modems enable a high bit rate, up to 60Mbit/s. Thus, the described communication environment offers a real Video-On-Demand system, where end-users may choose their dedicated programs from a catalogue, and ask for playing and pausing them, without useless bandwidth usage.

As an example, the described communication environment comprises set-top-boxes (hereinafter "STBs") with LAN connectivity (for "Local Area Network") constituting the IRDs, VDSL modems, and a PC server based on the Linux technology. Of course, an operating system other than Linux could be used.

Each of the IRDs comprise a requesting device, while the server comprises an associated transmitting device.

In variants:

- ADSL (for "Asymmetric Digital Subscriber Loop") modem technologies and multicasting are used instead of the present VDSL modem embodiments, 'x'DSL in general providing other possibilities (HDSL, 10MDSL, ADSL2...);

- USB (for "Universal Serial Buses") connectors instead of Ethernet links at the users side (other connection means, e.g. directly on the bus are also possible);

- and/or only one IRD is present at the user side, which is directly connected to the second VDSL modem.

A/ System description: requirements

The main requirement is to serve on request a video/audio stream to a user. The limit is fixed to up to 5 simultaneous streams according to the present embodiment. In other embodiments, there ay be a different limit, or no limit at all.

The end-users of the TV sets have notably the capability to choose one program from a catalogue, play it, stop it and pause/resume it.

A1/ Network

The VDSL modems, as the ADSL ones, use the actual user local telephone line. They have Ethernet or USB (for "Universal Serial Buses") connectors.

The transport format is an MPEG2 (for "Moving Picture Experts Group") SPTS (for "Simple Program Transport Stream"), several audio information sets being possibly transported for language purpose. In this way, the STBs have naturally the capability to play this type of stream easily.

At the server side, the PC is connected through an Ethernet 100BT link (for "100Base-T") to the modem.

At the user side, the STBs (up to four) are connected through an Ethernet 10BT link to the switch, which is a 100BT/10BT switch. Any of the users may then share its VDSL line for different usages, like a home PC, other room STBs, and so on. Also, by using the switch, the collision spaces are split, thus no collision propagation may occur.

In the illustrated example, a specific protocol above UDP (for "User Datagram Protocol") protocols stack is used. Advantages of that solution

(insofar as the links are fair enough), rely on the bandwidth delay introduced by the modem and on the current availability of STBs for which the Ethernet interface has only half-duplex capabilities; then, there is no need of a more complicated and less efficient protocol. Namely, experimentation shows that TCP (for "Transmission Control Protocol") protocols stack has real limitation with this network environment, even using the so-called "Window Scale Option".

A2/ Server

Based on Linux, it:

- includes a DHCP server (for "Dynamic Host Configuration Protocol") intended to setup the end-users IP (for "Internet Protocol") addresses;
- runs the application as a daemon;
- gives access to a catalogue of programs, each program being a file based on the server hard disk drive;
- streams the file based on the developed protocol;
- manages the user connection / disconnection states, as well as user dead state; a known port is used.

The server is advantageously in a central office, just linked to DSLAM (for "Digital Subscriber Line Access Multiplexer"). Its catalogue is then managed by operators in an efficient way (refresh method, cache algorithms...).

A3/ Set-top-boxes

They are able to:

- choose one program from a list;
- start that program by pressing a "Run program" button;
- pause/resume the program by pressing a "Pause/resume program" button;
- zap to another program;
- decode correctly and synchronize Audio and Video,
- accept intrinsic video bit rate from flow server up to 5 Mbit/s.

B/ The protocol

B1/ Requirements

The server and STBs are connected each other on TCP/IP LAN. In practice, one of the STBs connects socket to server and requests an audio/video file. The server then puts that file in a loop process only ended by a later STB request.

File transfer protocol, specifically designed, allows:

- query for available catalogue;
- connecting for stream transfer parameters setup;
- opening/closing a file;
- streaming an MPEG2-SPTS file, taking into account client data consumption;
- possibly indexing the file;
- checking for a client dead state.

The protocol is built to :

- transfer streamed data, thus no data recovery is needed;
- work on a fair network, i.e. without data loss, which is the case in the present system since no collision occurs because of the used modems and switch;
- work with long fat pipes, i.e. a lot of data stored either in the network, either in the client buffers.

This makes UDP quite appropriate as lower protocol.

The pipes and buffer involved in the streaming are represented on Figure 2, MLMP meaning "Main Level Main Profile".

The system may be seen as if a client (namely at one of the IRDs) requests data from a storage medium, which here is in a network. Thus, the client regulates the data streaming, according to the MPEG decoding processes.

Server computations

The protocol being based on UDP, and the server having big bandwidth capacity, the server has to take care to not overflow the client. Parameters are given by the client at a setup time to ensure this. These parameters are:

- client max supported bandwidth: CBW;
- client socket buffer size: CSOCKBUFSZ;
- and client keep alive message repeat period.

The server streams data according to these parameters; it computes the following value:

UNIT_SIZE = CSOCKBUFSZ / 8.

Then, it loops sending data putting a ceiling corresponding to that value (looping "UNIT_SIZE" by "UNIT_SIZE"), and inserting a delay in order to not override the CBW bit rate:

DELAY = UNIT_SIZE / CBW.

Client computations

Parameters are computed by the client in order to pause/resume without any hole in the needed bandwidth for streaming a file. These parameters are:

- FIFO high threshold <= CBW * RTT / 2

- FIFO low threshold >= CBW * RTT / 2

where "RTT" is the Round-Trip-Time, namely the delay between sending a packet and getting its acknowledgement. This time is measured by the client at the setup time, and may be re-evaluated in a periodic basis.

Socket buffers

These buffers are embedded with UDP/IP stack. They are large enough to not overrun the STB. The server computes its streaming burst according to the client socket buffer size.

FIFO

This buffer is used to deal with burst side-effects. The client starts decoding only when this FIFO has enough data. He pauses the server streaming when this FIFO is full, and resumes the server streaming when this FIFO is low.

MLMP MPEG buffers

These buffers are necessary to correctly decode an MPEG2 PES (for "Packet Elementary Stream") stream. The STB embedded audio/video decoder stack is used. The Main-Level / Main-Profile is implemented.

Because there is no encoding process at server side, the PCRs (for "Program Clock References") are not managed, but the local VCXO (for "Voltage Control Crystal Oscillator") is set to 27MHz. In any way, audio is synchronized on video using video and audio PTSs (for "Program Time Stamps"). The decoding processes is buffer-underflow tolerant, and does not reset or flush the MLMP buffers on errors.

B2/ Protocol stack

The protocol and streaming state transition diagrams are respectively represented on Figures 3 and Figure 4, from server side.

Protocol stack used is as follows:

Layer	Protocol
5/7 (application)	Application specific
	Socket
4 (transport)	UDP
3 (network)	IP
2 (link)	Ethernet
1 (physical)	Ethernet 802.3

Socket is open over UDP protocol.

B3/ Application protocol

Process is cut in the following steps:

- create socket;
- application layer process comprising:
    - initialize exchange : "Open Stream",
    - exchange : "Write/read Stream",
    - end exchange : "Close Stream";
- and close socket.

Packets built in application layer are messages.

All messages have a generic format :

- header :
    - message type (1 long)
    - up to four optional parameters (4 longs)
- data : a piece of the stream or empty for signalization messages.

All data types are network aligned, that is big-endian (MSB, for "Most Significant Bit").

a/ Connect socket

Server		STB
Socket SetOption Bind RecvFrom (IN port=1040, OUT ClientIpAddr)		Socket SetOption
	←	SendTo (IN portDest=1040,IN IpAddrDest)
Fork Socket SetOption Bind(IN port ephemeral, IN ClientIpAddr)		

IP addresses are constants.

b/ Initialize connection

Server		STB
Read(...)	←	Write(AL_CONNECT)
Write(AL_CONNECT_ACK)	→	Read(...) Compute RTT
Read(...)	←	Write(AL_CONNECTED, CBW, CSOCKBUFSZ, KEEPALIVE repeat period)

AL_CONNECT = 0x03 (1 long)	Unused (4 longs) (4 longs)
AL_CONNECT_ACK= 0x04 (1 long)	Unused (4 longs) (4 longs)
AL_CONNECTED = 0x05 (1 long)	Client max bitrate in Kbit/sec (1 long) Client buffer socket size in bytes (1 long) Client keep alive message repeat period in msec (1 long) Unused (1 long) (4 longs)

c/ Open/close a stream

Server		STB
Read(...) Open file	←	Write(AL_FILE_LIST)
Write(AL_FILE_LIS_ACK, DATA_PACKET_LEN)	→	Read(...)
Write(AL_FILE_LIST _DATA,...)	→	Read(...)
Wrtie(AL_FILE_LIST _DATA,...)	→	Read(...)
...	→	...
Read(...)	←	Write(AL_OPEN, file_index)
Read(...) Close file	←	Write(AL_CLOSE)

AL_FILE_OPEN = 0x01 (1 long)	File index in the catalogue (1 long) Unused (3 longs) (4 longs)
AL_CLOSE = 0x02 (1 long)	Unused (4 longs) (4 longs)
AL_FILE_LIST = 0x01	Unused (4 longs)

(1 long)	(4 longs)
AL_FILE_LIST_ACK= 0x04 (1 long)	Number of AL_FILE_LIST_DATA packets following in the stream (1 long) Unused (3 longs) (4 longs)
AL_FILE_LIST_DATA= 0x04 (1 long)	Catalogue description data, format to be defined (4 long) (4 longs)

d/ Streaming

Server		STB
Write(stream_data)	→	Read(…)
Write(stream_data)	→	Read(…)
…	…	…
Read(…) Restart timer alarm	←	Write(AL_KEEP_ALIVE)
Write(AL_KEEP_ALIVE_ACK) Restart timer alarm	→	Read(...) Compute RTT
Write(stream_data)	→	Read(…)
Write(stream_data)	→	Read(…)
…	…	…
Read(…) Pause streaming Wait for XON	←	Write(AL_XOFF)
Read(…) Resume streaming	←	Write(AL_XON)
Write(stream_data)	→	Read(…)
Write(stream_data)	→	Read(…)
…	…	…

AL_KEEP_ALIVE = 0x08 (1 long)	Unused (4 longs) (4 longs)
AL_KEEP_ALIVE_ACK= 0x0B (1 long)	Unused (4 longs) (4 longs)
AL_XOFF = 0x07 (1 long)	Unused (4 longs) (4 longs)
AL_XON= 0x06 (1 long)	Unused (4 longs) (4 longs)

e/ End exchange and close sockets

Server		STB
Alarm triggers because of no keep alive message	...	Exit, standby or crash
Close socket Exit process	...	

C/ Set-top-boxes

Once one of the STBs sends "AL_OPEN" message, it waits until the server sends stream data packets (reception is non blocking). The STB reads at much as possible data from the socket, release also the CPU (for "Central Processing Unit") for injection and decoding processes to be scheduled. One suitable algorithm consists in reading half buffer socket size at each schedule of a data pump, being part of the requesting device incorporated in the STB.

With presently available STB designs, the maximum measured sustained bit rate is around 6 Mbit/s, which is high enough for good MPEG2 video and audio quality.

The data pump gives directly the "Write" pointer of a FIFO (for "First In First Out" memory) and the contiguously available space to the socket buffer (this avoids intermediate copy from socket buffer to FIFO buffer).

"LenAsked" value is half FIFO size and avoids heavy scheduling works. After FIFO is fulfilled, LenAsked is the contiguously available size according to consumer pointer. An Injector being part of the requesting device is intended to manage it.

If the user zaps from one flow to another, the socket, the FIFO and the bit buffers may be flushed.

The server does not anticipate the read file. The read length and sent length are constant and smaller than internal socket buffer in order to avoid burst transfers: protocol also regulates transmit rate to consumer rate (FIFO managed by data pump on STB).

The IRD processing overview is explained below (see Figure 5), the server software analysis being made later. The various described elements are parts of the IRD requesting device.

- **Zapper:** gets user input selection (program identifier), runs the data pump, injector and starts AV decoders; if the user changes program, the zapper stops data pump and injector before running them again. The video decoder does not need to be stopped.

- **Ethernet driver:** interfaces TCP/IP stack to a physical Ethernet controller, which is for example the one commercialised as "CS8900A" by the Company Cirrus Logic. The rate is comprised between 3 Mbit/s and 6 Mbit/s. So, if the rate is 4 Mbit/s, one frame (1460 bytes) is processed in 2.86 ms.

- **TCP/IP stack:** TCP/UDP/IP stack over Ethernet, with socket non-blocking API (for "Application Programming Interface"). A DHCP client is embedded in order to get an IP address dynamically. The TCP/IP stack makes use of a socket buffer, having 64 kbytes max, and of a timer.

- **Data Pump:** uses a UDP port in order to get MPEG2-TS (for "Transport Stream") data from the server. In operation, the file identifier is specified by the user and transmitted directly to the server during connection. Once connected to the server, the data pump reads the available FIFO length and asks to the server a variable quantity of data as many times as necessary to have the FIFO nearly full. It allows also the user to zap thus to flush the FIFO. The flush method is also propagated to the injector.

- **FIFO** : X Mbytes FIFO (for example 1 Mbyte), used to regulate the decoding process and the streaming process, and to deal with network bursts. A FIFO manager manages two thresholds, the high triggering the injector process, and the low triggering the data pump process.

For transmissions, a unit size of 32 kbytes is for example chosen. With a unit size of 50 kbytes, a transmission takes 20 ms. Also, considering a rate of 4 Mbits/s, it takes 2 seconds to fill the FIFO, a full FIFO corresponding to 4 seconds decoding, and a half full FIFO to 1 second decoding.

- **Injector :** The notion of flow control between the PC server (producer) and the decoding process (consumer) is computed here: once started, the injector polls a video bit buffer level and injects via a DMA (for "Direct Memory Access") to the TSMUX (meaning "Transport Stream Multiplexer"), using SWTS input (for "Software Transport Stream"), a given quantity of FIFO data. The FIFO must then have sufficient data.

- **TSMux:** based on registers like those commercialised as "ST5516" by the Company ST Microelectronics, this interface permits injection of data from memory to a PTI (for "Programmable Transport Interface") in SWTS mode. It stores temporally data in its internal 32-byte FIFO and streams them out to the PTI. Speed of byte transfer can be adjusted. The pace rate is worth for example 40 Mbits/s.

- **AV stack:** audio and video decoding process; allows to setup the decoding processes, to setup the PID (for "Packet IDentifier") to decode. It allows to start the decoding processes, and to indicate Video bit buffer level (noted "VBBL") – the bit buffers are underflow tolerant.

- **AV synchronisation:** it is based on PTS values (for "Program Time Stamps") from Video and Audio. No PCR is managed so that the 27 MHz VCXO has a fixed command value. Video decoding process starts and initializes the STC value (for "System Time Clock") with the first Video PTS. Audio skips frame until its PTS value is near STC value.

C1/ Injector

The injector and its functions are now detailed, with reference to Figure 6.

Injection of data to SWTS input is paced using a "SWTS_REQ" signal that is routed through the PTI "DMA3" channel. That SWTS_REQ signal is asserted when the internal SWTS FIFO has room for at least 16 bytes. Each time the SWTS_REQ is low and there is data to be read from the X-Mbytes FIFO, a programmable number of bytes is transferred to the SWTS input. At the end of the complete transfer, an interrupt is set to signal injector task that data have been transferred.

In case of the SWTS entry, the destination pointer is fixed: that injection is processed like "CD_FIFO", SWTS register address replaces CD_FIFO address and no increment is programmed directly by "PTI_DMA3SETUP" register (so: "DMA3BLOCKMOVE = 0").

PTI version driver

The PTI driver is able to provide software support for DMA injection. It runs in "PTI3" hardware used in "PTI1" backwards compatible mode. In the presented embodiment, the only restriction is that interrupt mode is not allowed for signalling DMA transfer end: the call to the "pti_data_dma_synchronize" function blocks until the end of the DMA transfer.

Interconnecting PTI DMA3 with TSMUX

The "SWTS_REQ" signal is multiplexed to the "PTI NOT_CDREQ3" signal by configuring "CONFIG_CONTROL_A" register (namely: "PTIA_CDREQ3_MUXSEL[3:0] = TSSUB_SWTS_REQ").

The injector task needs to know the level of occupation of Audio and Video bit buffer before injecting data in the PTI. Supposing that each stream contains a video and an audio component and audio bit buffer capacity is enough sufficient to avoid overflow and underflow, monitoring only video bit buffer level may be acceptable.

To do this, the "VID_VBL" register is read, which gives the current level of Video bit buffer in units of 2 kbits (256 bytes). The size of free space in this buffer is then immediately deduced. The injector task starts DMA transfer from user FIFO to the PTI through the TSMUX, the size of transfer being:

min(VBB, Fifo available data)

Thus, the transfer is computed taking into account only VBB (for "Video Bit Buffer"). This implies that the ABB (for "Audio Bit Buffer") is large enough to not overflow.

For a flow at 4Mbits/s (500kbytes/s), the transfer of 50kbytes of data appends every 100 ms. According to SWTS pace rate fixed to 40 Mbits/s (5 Mbytes/s), the transfer duration takes at least 10 ms. So, that thread sleeps for 90 ms between each transfer.

C2/ Audio / Video stack

Audio/Video Stack has three main functions:

- configure demultiplexing, start audio and video decoders;
- manage synchronization between Audio and Video;
- and regulate the data streaming, getting Video bit buffer level, and supply it for Injector client.

It is not necessary to pause or freeze video, mute sound or display still picture.

The synchronization of Audio from Video decoders is made thanks to Audio PTS and Video PTS. The PCRs from flow are not managed, the VCXO is set to 27MHz.

STC value is set with first Video PTS. The synchronisation algorithm already exists in audio driver. The only need is to modify a function in order to set STC value with Video PTS. This can be processed as follows.

Watching video PTS is not aborted in the case an error occurs in the stream (STC/video PTS distance is updated every time a PTS occurs).

The stack is also tolerant regarding bit-buffers underflow, i.e. neither decoding process reset, nor bit-buffer reset.

The bit-buffers (VBB and ABB) are correctly sized for a PULL model, that is the decoder process regulates the data streaming. As concerns the VBB, it is sized for MLMP feature, around 356 Kbytes. To size the ABB, the ratio between the lowest video bit rate and the highest audio bit rate is computed. This ratio is applied as follows to size the ABB:

- Min video bit rate 1Mb/s,
- Max audio bit rate 448Kb/s,
- **ABB size = 458752 / 1048576 * MLMP VBB size.**

CLAIMS

1. Data requesting device through a network from a data server, comprising means for sending requests of determined data to the server and means for receiving streamed data from said server and for providing said data for them to be exploited,

characterized in that said data requesting device comprises also means for repeatedly determining successive required portions of said data as said data are exploited, said means for sending requests being intended to be triggered by said determining means as data are required, and said receiving means being intended to receive the streamed successive portions of said data.

2. Data requesting device according to claim 1, characterized in that the determining means comprise a data pump, intended to fill a storing space with said requested data as the data in that storing space are extracted for said exploitation.

3. Data requesting device according to any of claims 1 or 2, characterized in that said exploitation consists in decoding.

4. Decoder, characterized in that it comprises a data requesting device according to any of claims 1 to 3.

5. Data requesting process through a network from a data server, in which requests for determined data are sent to the server and corresponding data are received via streaming from said server,

characterized in that successive required portions of said data are repeatedly determined as said data are exploited, said requests being sent to said server as data are required according to said determined portions, and

said streamed successive portions of said data are received and provided to be exploited.

6. Data transmitting device comprising means for receiving data requests from clients and means for triggering streaming of said data to said clients,

characterized in that said receiving means are intended to repeatedly receive information on successive portions of said data and said triggering means are intended to provide said portions only and when said information is received.

7. Data transmitting process in which data requests are received from clients and streaming of said data to said clients is triggered,

characterized in that information on successive portions of said data is repeatedly received and said portions are provided only and when said information is received.

8. Computer program product, characterized in that it comprises instructions able to put in practice any of the processes of claims 5 and 7 when it is operated in a computer.

ABSTRACT

The present invention concerns data requesting and transmitting devices and processes.

It relates notably to a data requesting device through a network from a data server, comprising means for sending requests of determined data to the server and means for receiving streamed data from that server and for providing those data for them to be exploited.

This data requesting device comprises also means for repeatedly determining successive required portions of those data as those data are exploited, the means for sending requests being intended to be triggered by the determining means as data are required, and the receiving means being intended to receive the streamed successive portions of those data.

FIG.5

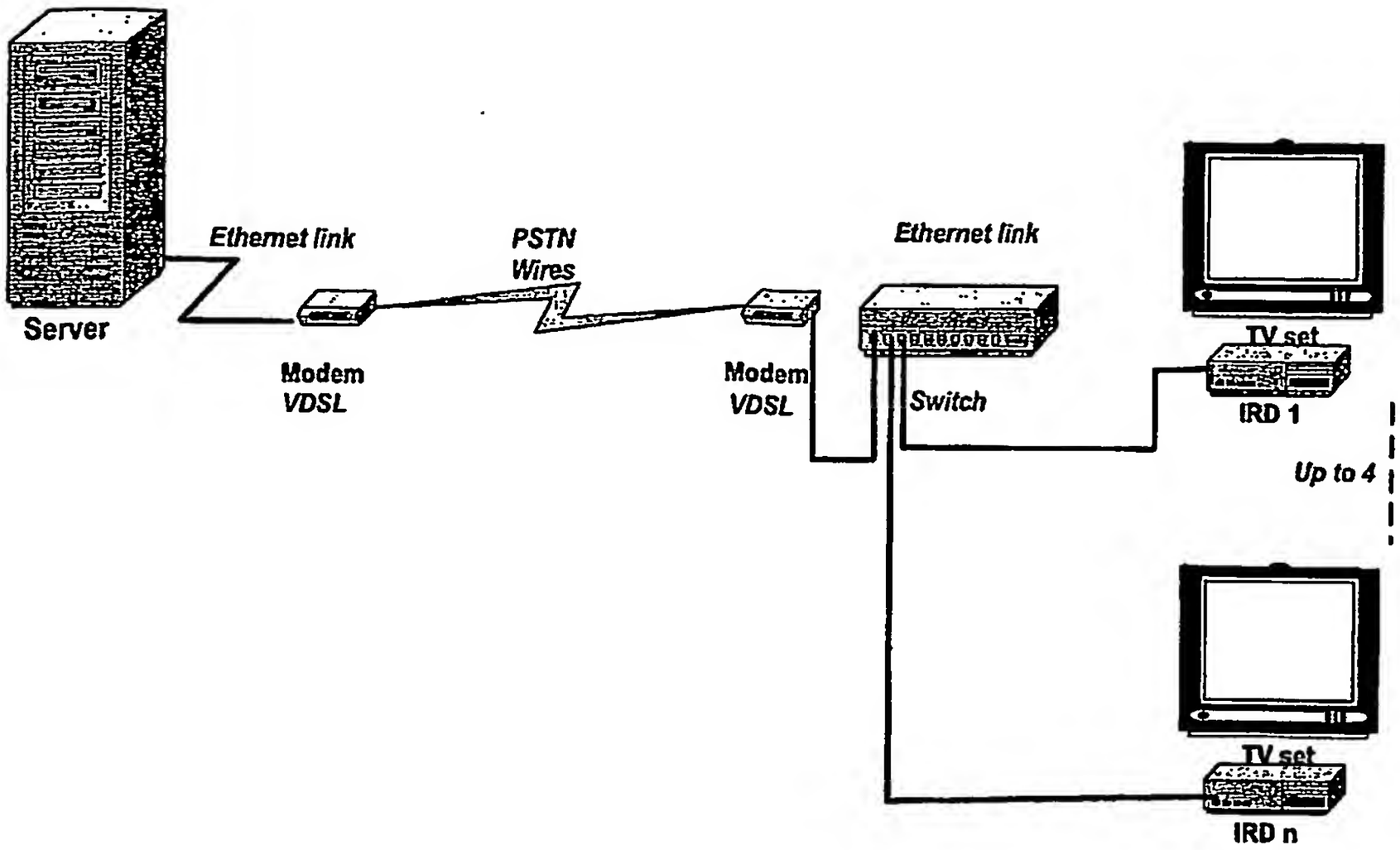
Server
Ethernet link
Modem
VDSL
PSTN
Wires
Modem
VDSL
Ethernet link
Switch
TV set
IRD 1
Up to 4
TV set
IRD n

FIG. 1

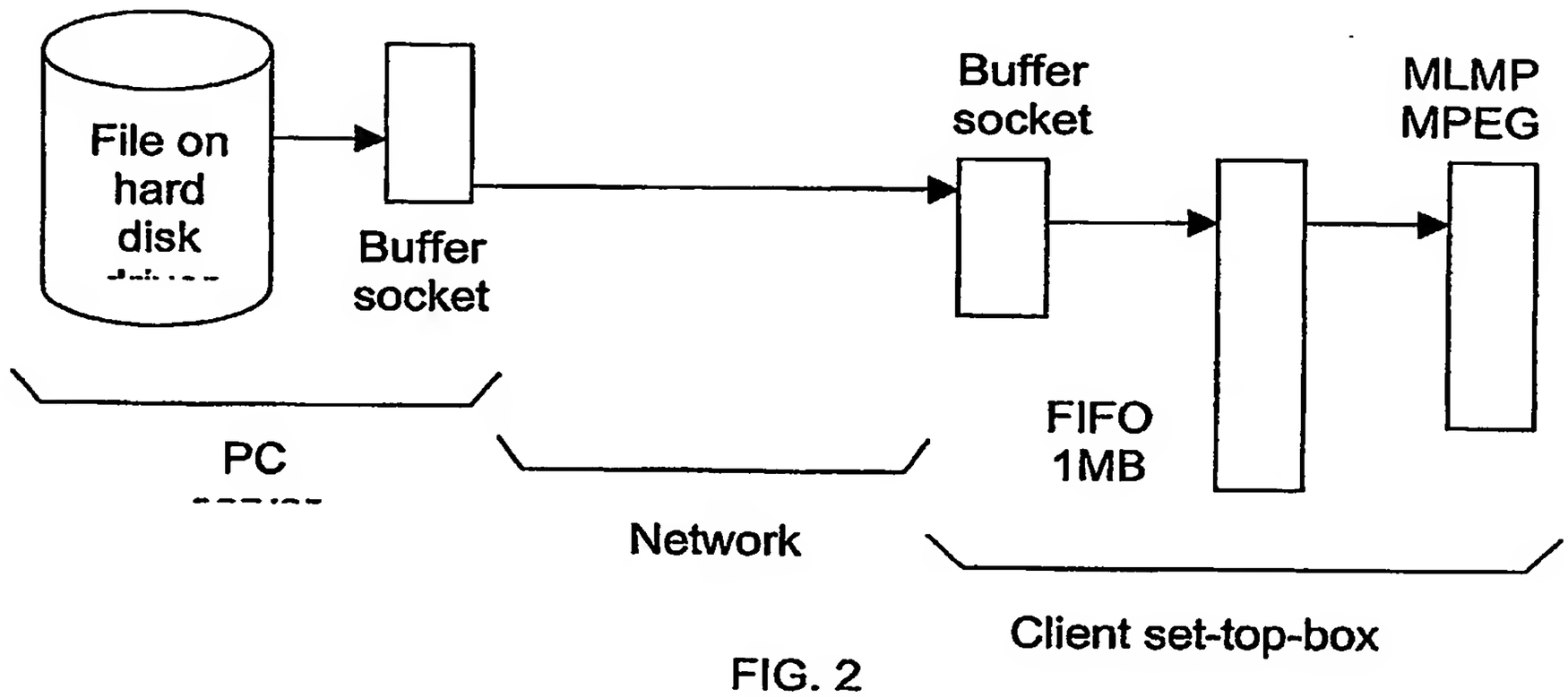
File on
hard
disk
Buffer
socket
Buffer
socket
MLMP
MPEG
FIFO
1MB
PC
Network
Client set-top-box

FIG. 2

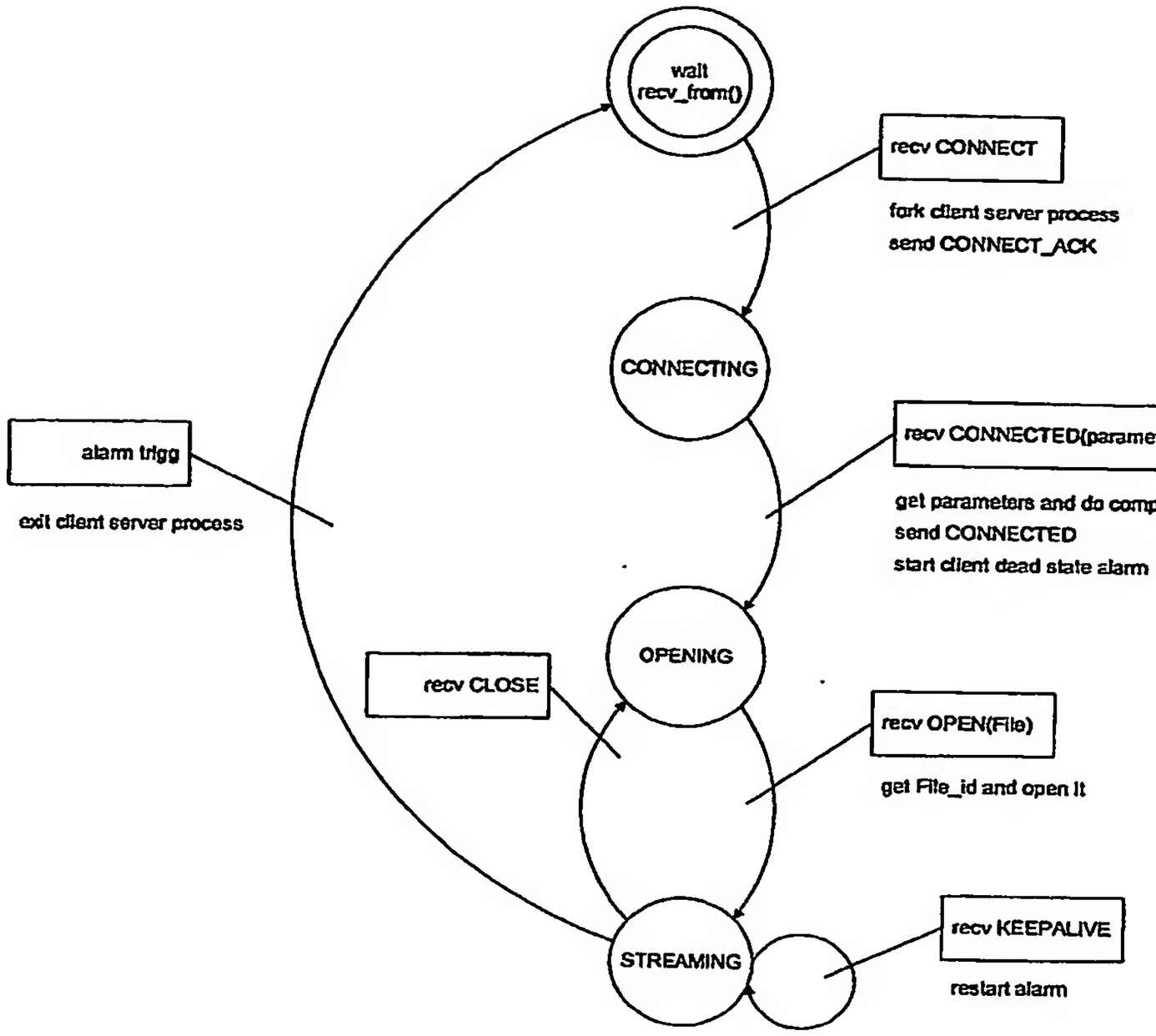
wait
recv_from()
recv CONNECT
fork client server process
send CONNECT_ACK
CONNECTING
recv CONNECTED(parameters)
get parameters and do computation
send CONNECTED
start client dead state alarm
alarm trigg
exit client server process
OPENING
recv CLOSE
recv OPEN(File)
get File_id and open it
STREAMING
recv KEEPALIVE
restart alarm

FIG. 3

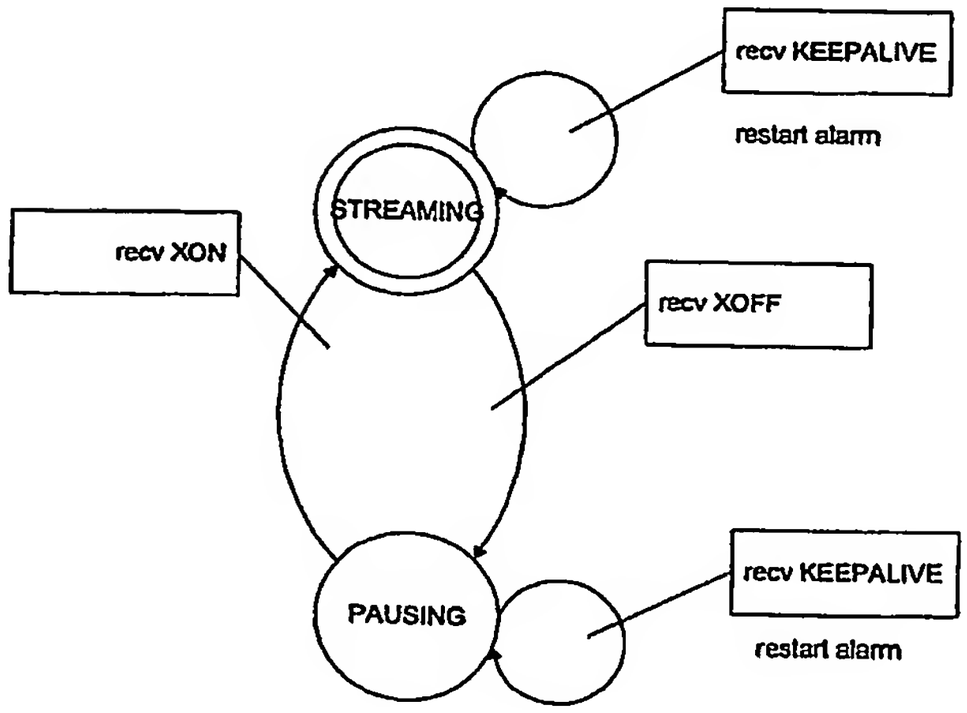
recv KEEPALIVE
restart alarm
STREAMING
recv XON
recv XOFF
PAUSING
recv KEEPALIVE
restart alarm

FIG. 4

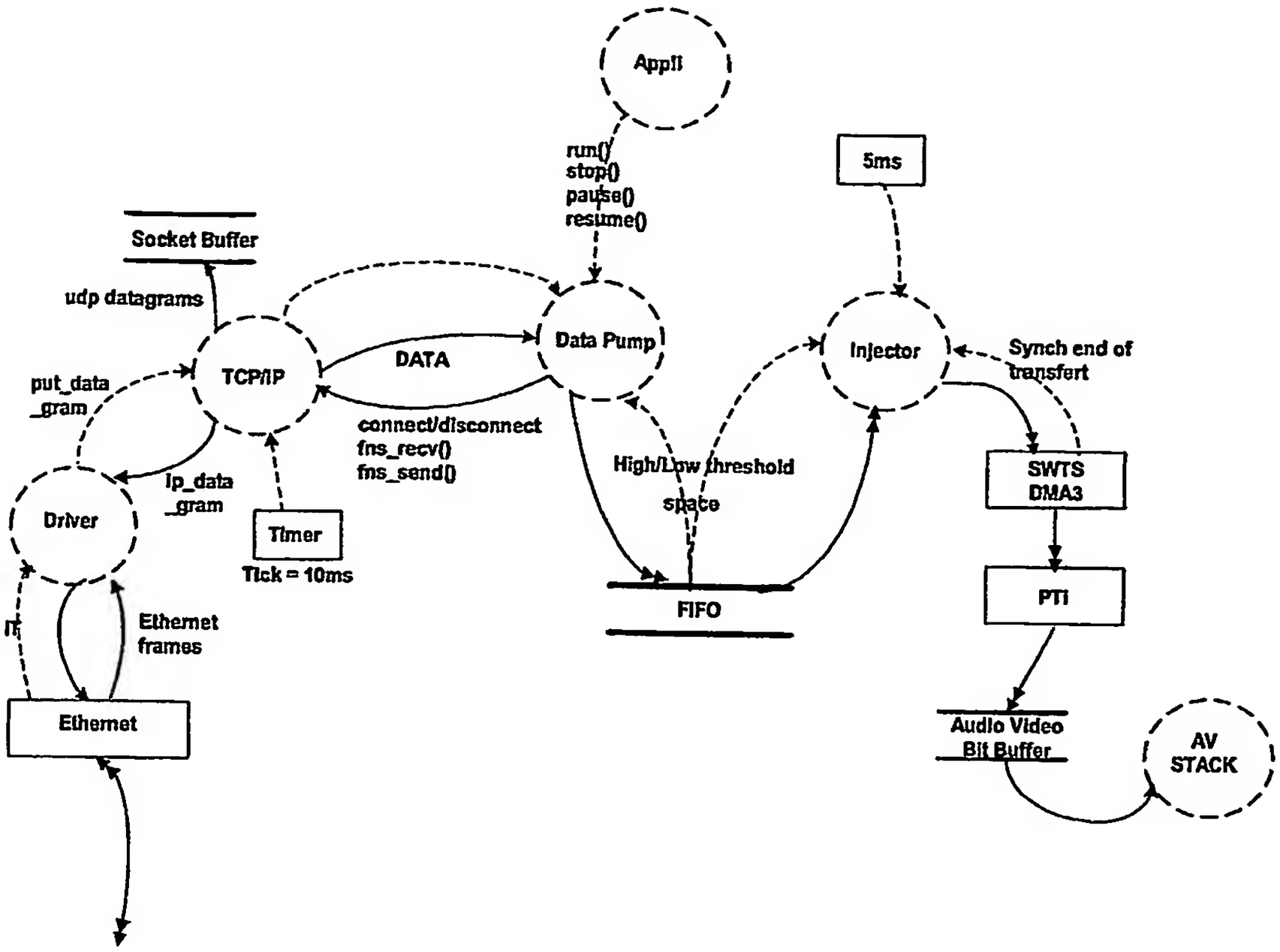
Appli
run()
stop()
pause()
resume()
5ms
Socket Buffer
udp datagrams
Data Pump
Injector
Synch end of transfert
put_data _gram
TCP/IP
DATA
connect/disconnect
fns_recv()
fns_send()
High/Low threshold
space
SWTS DMA3
Ip_data _gram
Driver
Timer
Tick = 10ms
FIFO
PTI
Ethernet frames
Ethernet
Audio Video Bit Buffer
AV STACK

FIG. 5

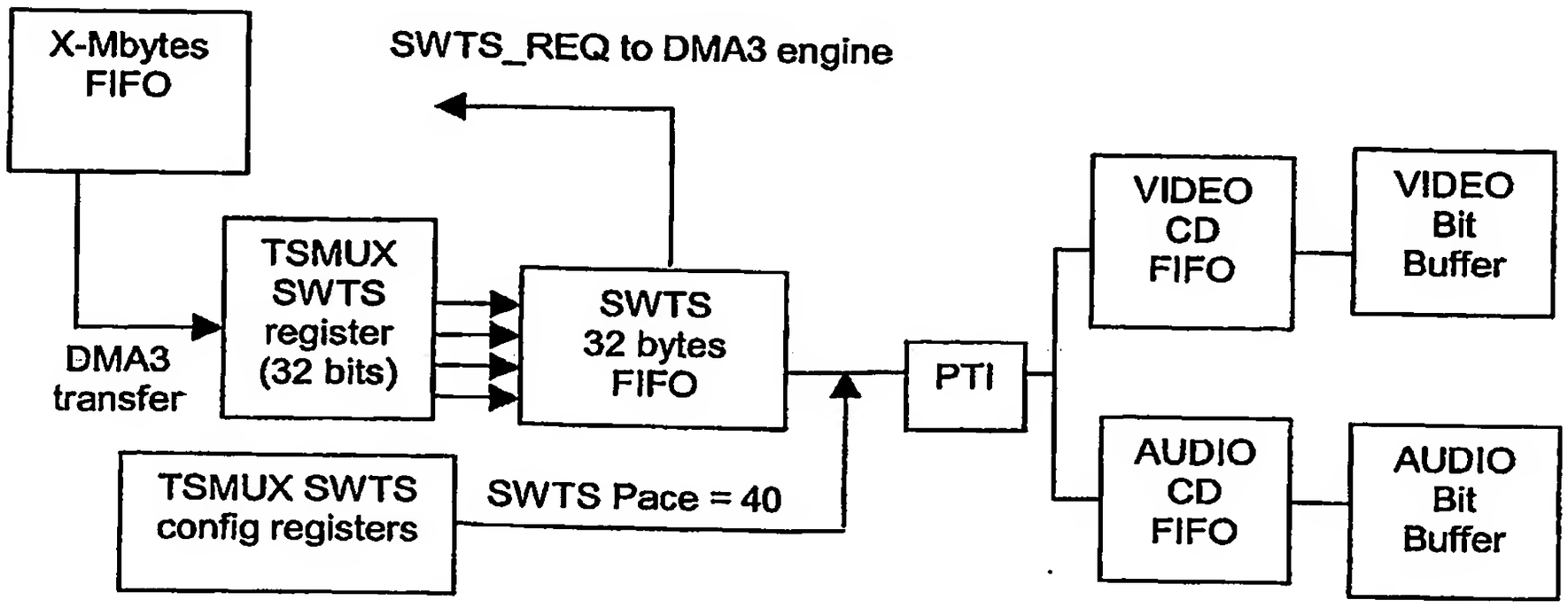
X-Mbytes FIFO
SWTS_REQ to DMA3 engine
TSMUX SWTS register (32 bits)
SWTS 32 bytes FIFO
VIDEO CD FIFO
VIDEO Bit Buffer
DMA3 transfer
PTI
AUDIO CD FIFO
AUDIO Bit Buffer
TSMUX SWTS config registers
SWTS Pace = 40

FIG. 6